

UNITED STATES PATENT APPLICATION

CACHE INTEGRITY APPARATUS, SYSTEMS, AND METHODS

INVENTORS

Robert J. Royer, Jr.

Richard L. Coulson

Schwegman, Lundberg, Woessner & Kluth, P.A.
1600 TCF Tower
121 South Eighth Street
Minneapolis, MN 55402
ATTORNEY DOCKET SLWK 884.905US1
Client Reference P16256

CACHE INTEGRITY APPARATUS, SYSTEMS, AND METHODS

Technical Field

Various embodiments described herein relate generally to apparatus,
5 systems, and methods used to maintain the integrity of data stored in a memory,
including cached data.

Background Information

A computer memory cache may operate using volatile or non-volatile
10 memory. When volatile memory is used, caching policies usually (i.e., since
the cache state is reset during each boot) include a mechanism to flush the
cache before associated cache driver software acts to access the cache.
Access requests to the cached medium that are made prior to loading and
executing the driver software can thus be prevented from corrupting cached
15 data.

While it is can be advantageous to use non-volatile memory in a
cache (e.g., the cache state can be maintained during system shutdown or a
power failure), implementation problems may arise. For example, write
requests associated with data stored in a non-volatile cache that occur prior
20 to loading the cache driver may result in data corruption once the driver
boots. Invalidating the non-volatile cache after the driver begins execution
may avoid data corruption, but the advantage of a persistent cached data
state is lost.

Brief Description of the Drawings

FIG. 1 is a block diagram of an apparatus, a system, and an article according
to various embodiments; and

FIG. 2 is a flow chart illustrating several methods according to various
embodiments.

Detailed Description

In the following detailed description of various embodiments, reference is made to the accompanying drawings that form a part hereof, and in which are shown by way of illustration, and not of limitation, specific embodiments in which the subject matter may be practiced. The embodiments illustrated are described in sufficient detail to enable those skilled in the art to practice the teachings disclosed herein. Other embodiments may be utilized and derived therefrom, such that structural and logical substitutions and changes may be made without departing from the scope of this disclosure. The following detailed description, therefore, is not to be taken in a limiting sense, and the scope of various embodiments is defined only by the appended claims, along with the full range of equivalents to which such claims are entitled.

In some embodiments, a non-volatile cache may be implemented by creating a device option read only memory (ROM or OROM), or modifying the related computer basic input-output system (BIOS) to trap software interrupts (e.g., the BIOS Int13h interface of Intel[®] x86-compatible machines) associated with disk access requests. Disk write activity can be detected in this manner, and associated addresses can be stored in a log for all modified data addresses (e.g., logical block addresses, or LBAs). This "write log" can be stored in a non-volatile medium, along with the cache itself. If the write log does not have enough space to record all disk write activity prior to loading the operating system (OS) drivers (e.g., the OS storage subsystem drivers), a flag may be set to indicate the overrun condition. The flag may be included in the log, or stored in another memory, which may be volatile or nonvolatile.

Once the OS storage subsystem drivers load, they may query the write log and invalidate or update any data in the cache that has been changed, as indicated by the write log. If the write log overrun flag is set, the driver can update all data in the cache from currently available disk data, or invalidate the entire cache, so that no data loss will occur, since the disk will always have the correct version of the data stored. After the driver successfully flushes all data associated with the addresses in

the write log, the log may be cleared to an empty state. If the cache policy permits dirty data (i.e., writeback policies), the option read only memory (OROM) may operate to flush all dirty data before allowing the system to boot.

FIG. 1 is a block diagram of apparatus 100, systems 110, and an article 112 according to various embodiments. The apparatus 100 may comprise a cache 114, which may be volatile or non-volatile, and a memory 118. The memory 118, which may also be volatile or non-volatile, can be used to store one or more addresses 122 associated with write operations to another memory 130 (cached by the cache 114) that occur prior to loading and/or executing an OS cache driver (not shown). The addresses 122 may be stored in the memory 118 in the form of a log.

If interrupts are trapped to detect write operations (e.g., the BIOS Int13h interrupt), the apparatus 100 may also include an interrupt module 134 to receive an interrupt request 136 associated with write operations to the memory 130. The addresses 122 may be LBAs, for example, and the interrupt request 136 may be a hardware interrupt request or a software interrupt request, perhaps including or associated with the occurrence of a BIOS Int13h interrupt.

The memories 114 and 130 may comprise any type of memory, including random access memory (RAM), polymer ferroelectric RAM (PFRAM), various kinds of programmable read only memory (PROM), magnetic or optical disks, etc. If the memory 118 is overrun by recording a number of addresses 122 that is larger than the log size, a flag 138 may be set to indicate the overrun condition.

For the purposes of this document, a memory is a volatile memory if the content stored therein is typically not retained when power is removed and re-applied to an associated processor. Non-volatile memory is any kind of memory that has the property of retaining content for a selected amount of time (e.g., several seconds or more) even when power applied to an associated microprocessor is cycled. Non-volatile memory includes, for example, battery-backed RAM, RAM backed by FLASH memory, and battery-backed RAM plus some other type of non-volatile storage, including a disk drive.

Other embodiments are possible. For example, a system 110 may comprise the apparatus 100, as described above, in combination with other elements. In an embodiment, the system 110 may comprise a processor 140 coupled to the apparatus 100, a disk drive 141 (typically comprising some form of non-volatile memory), and a display 144 coupled to the processor 140. The display 144 may be used to display a graphical user interface (GUI) 148.

The system 110 may include an interrupt module 134 to receive an interrupt request 136 associated with write operations to the memory 130. The module 134 may be included in a device option memory 152, or it may be implemented as part of a BIOS module 156. As noted previously, the memory 118 may be volatile or non-volatile, and it may be used to store a log having one or more memory addresses 122 associated with write operations to the memory 130.

The apparatus 100, systems 110, article 112, cache 114, memory 118, addresses 122, memory 130, interrupt module 134, interrupt request 136, flag 138, processor 140, disk drive 141, display 144, GUI 148, device option memory 152, and BIOS module 156 may all be characterized as “modules” herein. Such modules may include hardware circuitry and/or one or more processors and/or memory circuits, software program modules, including objects and collections of objects, and/or firmware, and combinations thereof, as desired by the architect of the apparatus 100, systems 110, and article 112, and as appropriate for particular implementations of various embodiments.

It should also be understood that the apparatus and systems of various embodiments can be used in applications other than for computers, and other than for systems that include displays and graphical user interfaces, and thus, various embodiments are not to be so limited. The illustrations of an apparatus 100 and a system 110 are intended to provide a general understanding of the structure of various embodiments, and they are not intended to serve as a complete description of all the elements and features of apparatus and systems that might make use of the structures described herein.

Applications that may include the novel apparatus and systems of various embodiments include electronic circuitry used in high-speed computers, communication and signal processing circuitry, modems, processor modules, embedded processors, and application-specific modules, including multilayer, multi-
5 chip modules. Such apparatus and systems may further be included as sub-components within a variety of electronic systems, such as televisions, cellular telephones, personal computers, workstations, radios, video players, vehicles, and others. Many different embodiments may be realized.

For example, FIG. 2 is a flow chart illustrating several methods according to
10 various embodiments. A method 211 may (optionally) begin with hardware initialization (e.g., the application of power to a processor) at block 213, and trapping an interrupt request, including a hardware or software interrupt request, at block 215.

The method 211 may then continue with detecting a write operation to a
15 memory cached by a volatile or non-volatile cache at block 217, and then recording the address of the write operation prior to loading and/or executing an operating system cache driver at block 221. Thus, detecting the write operation at block 215 may include trapping an interrupt request at block 213. This activity may be followed with issuing a write to memory at block 223.

20 Recording the address of the write operation at block 221 may include recording the address in a log (which may be stored in a volatile or non-volatile memory, including a memory comprising at least one of a random access memory (RAM), including a static RAM (SRAM), a dynamic RAM (DRAM), a flash memory, and a polymer ferroelectric RAM (PFRAM) at block 225.

25 Since the log is typically of a limited size, it may be possible to overrun the log by recording write operations to a number of addresses that is larger than the log size. If the log is determined to be overrun at block 231, then the method 211 may include setting a flag to indicate an overrun of the log at block 235. Otherwise, the method may continue with issuing a write to memory at block 223, and determining
30 whether an OS cache driver is loaded and/or executing at block 239. If the driver is

not yet loaded and/or executing, other write operations to the cached memory may be detected at block 215. Otherwise, the method may continue with block 245.

The method 211 may thus include modifying data corresponding to the address of the write operation at block 245. If the log overrun flag is not determined to be set at block 249, modifying the data corresponding to the address of the write operation may comprise updating the data corresponding to the address of the write operation at block 255. If the log overrun flag is determined to be set at block 249, then modifying the data corresponding to the address of the write operation may comprise invalidating the data corresponding to the address of the write operation at block 259, which may include invalidating the cache if the flag is set. Alternatively, all of the data in the cache may be updated at block 259.

It should be noted that the methods described herein do not have to be executed in the order described, or in any particular order. Moreover, various activities described with respect to the methods identified herein can be executed in serial or parallel fashion. Information, including parameters, commands, operands, and other data, can be sent and received in the form of one or more carrier waves.

Upon reading and comprehending the content of this disclosure, one of ordinary skill in the art will understand the manner in which a software program can be launched from a computer readable medium in a computer-based system to execute the functions defined in the software program. One of ordinary skill in the art will further understand the various programming languages that may be employed to create one or more software programs designed to implement and perform the methods disclosed herein. The programs may be structured in an object-orientated format using an object-oriented language such as Java, Smalltalk, or C++. Alternatively, the programs can be structured in a procedure-orientated format using a procedural language, such as COBOL, assembly, or C. The software components may communicate using any of a number of mechanisms well-known to those skilled in the art, such as application program interfaces or interprocess communication techniques, including remote procedure calls. The teachings of various embodiments of the present invention are not limited to any particular

programming language or environment, including Hypertext Markup Language (HTML) and Extensible Markup Language (XML).

FIG. 1 also illustrates an article 112 including a machine-accessible medium according to various embodiments. Thus, it is now apparent that another

5 embodiment may include an article 112 such as a computer, a memory system, a magnetic or optical disk, some other storage device, and/or any type of electronic device or system, comprising a machine-accessible medium such as a memory 130 (e.g., a memory including an electrical, optical, or electromagnetic conductor) having associated data 162 (e.g., computer program instructions), which when
10 accessed, results in a machine performing such actions as recording addresses of write operations to a memory cached by a non-volatile cache prior to executing an operating system cache driver, which may include recording the address of the write operation in a log.

The log may be included in a volatile or a non-volatile memory. If the log is
15 overrun, the machine may act to set a flag, indicating the overrun condition. If the flag is set, the machine may also act to invalidate the cache, or update all of the data in the cache.

The ability to use a non-volatile cache can add versatility to the operation of a computer system, since the state of the cache can be maintained during system
20 shutdown or power failure. Recording the occurrence of write requests associated with data stored in the non-volatile cache that occur prior to loading the cache driver may thus prevent data corruption after the driver begins execution, such that the cache data need not be invalidated.

Although specific embodiments have been illustrated and described herein, it
25 should be appreciated that any arrangement calculated to achieve the same purpose may be substituted for the specific embodiments shown. This disclosure is intended to cover any and all adaptations or variations of various embodiments of the invention. It is to be understood that the above description has been made in an illustrative fashion, and not a restrictive one. Combinations of the above

embodiments, and other embodiments not specifically described herein, will be apparent to those of skill in the art upon reviewing the above description.

It is emphasized that the Abstract of the Disclosure is provided to comply with 37 C.F.R. §1.72(b), requiring an abstract that will allow the reader to quickly
5 ascertain the nature of the technical disclosure. It is submitted with the understanding that it will not be used to interpret or limit the scope or meaning of the claims. In addition, in the foregoing Detailed Description, it can be seen that various features are grouped together in a single embodiment for the purpose of streamlining the disclosure. This method of disclosure is not to be interpreted as
10 reflecting an intention that the claimed embodiments of the invention require more features than are expressly recited in each claim. Rather, as the following claims reflect, inventive subject matter lies in less than all features of a single disclosed embodiment. Thus the following claims are hereby incorporated into the Detailed Description, with each claim standing on its own as a separate preferred
15 embodiment.